AI Assisted Coding
Lab Assignment 6.5

Name : K. Charan Yadav

Hall Ticket no : 2303A52367

Batch No : 20

Task -1:

Prompt: Generate Python code to check voting eligibility based on age and citizenship

OUTPUT :

Task 2:

Prompt: Generate Python code to count vowels and consonants in a string using a loop

```
18    # Task -2 : Vowel and Consonant Counter
19    def count_vowels_and_consonants(text):
20        vowels = "aeiouAEIOU"
21        vowel_count = 0
22        consonant_count = 0
23        for char in text:
24            if char.isalpha():
25                if char in vowels:
26                    vowel_count += 1
27                else:
28                    consonant_count += 1
29        return vowel_count, consonant_count
30    # Test cases
31    test_string = "Hello World"
32    vowels, consonants = count_vowels_and_consonants(test_string)
33    print(f"String: '{test_string}'")
34    print(f"Vowels: {vowels}")
35    print(f"Consonants: {consonants}")
36    test_string2 = "Python Programming"
37    vowels2, consonants2 = count_vowels_and_consonants(test_string2)
38    print(f"\nString: '{test_string2}'")
39    print(f"Vowels: {vowels2}")
40    print(f"Consonants: {consonants2}")
```

Output:

```
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding>
ktop/Ai-Assisted Coding/Assignment-6.5.py"
String: 'Hello World'
Vowels: 3
Consonants: 7

String: 'Python Programming'
Vowels: 4
Consonants: 13
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding>
```

Task 3:

Prompt : Generate a Python program for a library management system using classes, loops, and conditional statements

```python
# Task -3 : Simple Library Management System
class Book:
    def __init__(self, book_id, title, author, available=True):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.available = available
    def __str__(self):
        status = "Available" if self.available else "Checked Out"
        return f"ID: {self.book_id}, Title: {self.title}, Author: {self.author}, Status: {status}"
class Library:
    def __init__(self):
        self.books = []
    def add_book(self, book):
        self.books.append(book)
        print(f"√ Book '{book.title}' added to library")
    def checkout_book(self, book_id):
        for book in self.books:
            if book.book_id == book_id:
                if book.available:
                    book.available = False
                    print(f"√ '{book.title}' checked out successfully")
                    return
                else:
                    print(f"X '{book.title}' is already checked out")
                    return
        print(f"X Book with ID {book_id} not found")
    def return_book(self, book_id):
        for book in self.books:
            if book.book_id == book_id:
                if not book.available:
                    book.available = True
                    print(f"√ '{book.title}' returned successfully")
                    return
                else:
                    print(f"X '{book.title}' is already available")
                    return
        print(f"X Book with ID {book_id} not found")
    def display_all_books(self):
        if not self.books:
            print("Library is empty")
            return
        print("\n--- Library Books ---")
        for book in self.books:
            print(book)
# Test the library system
library = Library()
library.add_book(Book(1, "Python Basics", "John Doe"))
library.add_book(Book(2, "Data Science", "Jane Smith"))
library.add_book(Book(3, "Web Development", "Mike Johnson"))
library.display_all_books()
library.checkout_book(1)
library.checkout_book(1)
library.return_book(1)
library.display_all_books()
```

Output :

```
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding> & C:/Users/chara/A
ent-6.5.py"
√ Book 'Python Basics' added to library
√ Book 'Data Science' added to library
√ Book 'Web Development' added to library

--- Library Books ---
ID: 1, Title: Python Basics, Author: John Doe, Status: Available
ID: 2, Title: Data Science, Author: Jane Smith, Status: Available
ID: 3, Title: Web Development, Author: Mike Johnson, Status: Available
√ 'Python Basics' checked out successfully
X 'Python Basics' is already checked out
√ 'Python Basics' returned successfully

--- Library Books ---
ID: 1, Title: Python Basics, Author: John Doe, Status: Available
ID: 2, Title: Data Science, Author: Jane Smith, Status: Available
ID: 3, Title: Web Development, Author: Mike Johnson, Status: Available
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding>
```

Task 4 :

Prompt : Generate a Python class to mark and display student attendance using loops."

Expected Output:

• AI-generated attendance logic.

• Correct display of attendance.

• Test cases

```python
# Task -4 : Student Attendance Tracker
class Student:
    def __init__(self, student_id, name):
        self.student_id = student_id
        self.name = name
        self.attendance = []
    def mark_attendance(self, date, status):
        self.attendance.append({"date": date, "status": status})
        print(f"✓ Attendance marked for {self.name} on {date}: {status}")
    def get_attendance_percentage(self):
        if not self.attendance:
            return 0
        present = sum(1 for record in self.attendance if record["status"].lower() == "present")
        return (present / len(self.attendance)) * 100
class AttendanceTracker:
    def __init__(self):
        self.students = []
    def add_student(self, student):
        self.students.append(student)
        print(f"✓ Student '{student.name}' added to tracker")
    def display_attendance(self):
        if not self.students:
            print("No students in tracker")
            return
        print("\n--- Attendance Report ---")
        for student in self.students:
            print(f"\nStudent: {student.name} (ID: {student.student_id})")
            for record in student.attendance:
                print(f"  {record['date']}: {record['status']}")
            print(f"  Attendance: {student.get_attendance_percentage():.1f}%")
```

```python
# Test cases
tracker = AttendanceTracker()
student1 = Student(101, "Alice")
student2 = Student(102, "Bob")
tracker.add_student(student1)
tracker.add_student(student2)
for date in ["2024-01-01", "2024-01-02", "2024-01-03"]:
    student1.mark_attendance(date, "Present")
    student2.mark_attendance(date, "Present")
student1.mark_attendance("2024-01-04", "Absent")
student2.mark_attendance("2024-01-04", "Present")
tracker.display_attendance()
```

Output :

```
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding> &
ent-6.5.py"
✓ Student 'Alice' added to tracker
✓ Student 'Bob' added to tracker
✓ Attendance marked for Alice on 2024-01-01: Present
✓ Attendance marked for Bob on 2024-01-01: Present
✓ Attendance marked for Alice on 2024-01-02: Present
✓ Attendance marked for Bob on 2024-01-02: Present
✓ Attendance marked for Alice on 2024-01-03: Present
✓ Attendance marked for Bob on 2024-01-03: Present
✓ Attendance marked for Alice on 2024-01-04: Absent
✓ Attendance marked for Bob on 2024-01-04: Present

--- Attendance Report ---

Student: Alice (ID: 101)
   2024-01-01: Present
   2024-01-02: Present
   2024-01-03: Present
   2024-01-04: Absent
   Attendance: 75.0%

Student: Bob (ID: 102)
   2024-01-01: Present
   2024-01-02: Present
   2024-01-03: Present
   2024-01-04: Present
   Attendance: 100.0%
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding>
```

Task 5 :

Prompt : Generate a Python program using loops and conditionals to simulate an ATM menu.

```python
141    # Task - 5 : ATM Simulation
142    class ATMSimulation:
143        def __init__(self, balance=1000):
144            self.balance = balance
145        def display_menu(self):
146            print("\n--- ATM Menu ---")
147            print("1. Check Balance")
148            print("2. Withdraw Money")
149            print("3. Deposit Money")
150            print("4. Exit")
151        def check_balance(self):
152            print(f"✓ Current Balance: ${self.balance:.2f}")
153        def withdraw_money(self):
154            try:
155                amount = float(input("Enter amount to withdraw: $"))
156                if amount <= 0:
157                    print("X Amount must be greater than zero")
158                elif amount > self.balance:
159                    print(f"X Insufficient funds. Available balance: ${self.balance:.2f}")
160                else:
161                    self.balance -= amount
162                    print(f"✓ Successfully withdrawn ${amount:.2f}")
163                    print(f"✓ Remaining balance: ${self.balance:.2f}")
164            except ValueError:
165                print("X Invalid input. Please enter a valid number")
166        def deposit_money(self):
167            try:
168                amount = float(input("Enter amount to deposit: $"))
169                if amount <= 0:
170                    print("X Amount must be greater than zero")
171                else:
172                    self.balance += amount
173                    print(f"✓ Successfully deposited ${amount:.2f}")
174                    print(f"✓ New balance: ${self.balance:.2f}")
175            except ValueError:
176                print("X Invalid input. Please enter a valid number")
```

```
177     def run(self):
178         print("✓ Welcome to ATM Simulation")
179         while True:
180             self.display_menu()
181             choice = input("Select an option (1-4): ")
182             if choice == "1":
183                 self.check_balance()
184             elif choice == "2":
185                 self.withdraw_money()
186             elif choice == "3":
187                 self.deposit_money()
188             elif choice == "4":
189                 print("✓ Thank you for using ATM. Goodbye!")
190                 break
191             else:
192                 print("X Invalid option. Please select 1-4")
193     # Test the ATM system
194     atm = ATMSimulation(1000)
195     atm.run()
```

Output :

```
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding>
ent-6.5.py"
✓ Welcome to ATM Simulation

--- ATM Menu ---
1. Check Balance
2. Withdraw Money
3. Deposit Money
4. Exit
Select an option (1-4):
```

Observation:

➢ The ATM simulation code provides a simple command-line interface or users to interact with an ATM system.

➢ It includes functionalities to check balance, withdraw money, deposit money, and exit the system.

➢ The code handles invalid inputs gracefully and provides appropriate feedback to the user.

➢ Overall, it demonstrates basic programming concepts such as classes, methods, and user input handling effectively.